Παναγιώτα Φυτάλη - p3200215

Σοφία Βέργη - p3200018

Σαραντία Μουρελάτου – 3200106

Τεχνητή Νοημοσύνη

1η Εργασία

Reversi (Othello)

Τρόπο χρήσης – Δυνατότητες – Μέθοδοι Τεχνητής Νοημοσύνης

Το Reversi(Othello) είναι ένα παιχνίδι αντιπάλων , για τα οποία γνωρίζουμε ότι στόχος κάθε παίκτη δεν είναι μόνο να κάνει κάποια κίνηση που ωφελεί τον ίδιο αλλά να επιλέξει την κίνηση, η οποία πέραν του πρώτου προκαλεί τον αποκλεισμό καλών, ή ορισμένων, κινήσεων για τον αντίπαλο, το οποίο τελικά εξυπηρετεί στην νίκη του παιχνιδιού και όχι στο άμεσο προβάδισμα το οποίο μπορεί να είχε η επιλογή μίας άλλης κίνησης. Με άλλα λόγια σε τέτοιου είδους παιχνίδια είναι σημαντικό να λαμβάνουμε υπόψη το σύνολο του παιχνιδιού και πως κάθε κίνηση εξυπηρετεί, στο τέλος, για την νίκη του, για να απλοποιηθούν τέτοια παιχνίδια είναι σημαντικό να ορίσουμε ότι θα υπάρχουν μόνο 2 αντίπαλοι, ότι οι στρατηγικές/κανόνες που καλύπτουμε είναι μοναδικές, δηλαδή το περιβάλλον που διαδραματίζεται το παιχνίδι , καθώς και οι συνέπειες κάθε κίνησης θεωρούνται πλήρως γνωστά. Τέτοια παιχνίδια όταν έχουν αντιπάλους δύο ανθρώπους δεν μπορούν να στηριχθούν στα παραπάνω, λόγω της φύσης του ανθρώπου, να μην γνωρίζει τους κανόνες (στρατηγικές όπως ονομάζονται) που θα έπρεπε να ακολουθήσει για μία «σίγουρη» κίνηση, ή να τους γνωρίζει αλλά να μην μπορεί να τους αναγνωρίσει κατά την διάρκεια του παιχνιδιού ή απλώς να κάνει κάποιο ανθρώπινο λάθος, αναφερόμαστε σε δύο πρόσωπα που δεν αποτελούν καλή αρχή μελέτης γιατί βασίζονται σε μεγάλο βαθμό στον παράγοντα της τύχης. Αν αναφερόμασταν σε ένα ιδανικό περιβάλλον, όπου οι παίκτες γνωρίζουν και ενδιαφέρονται για το τι κίνηση να κάνουν ώστε να επιφέρει τελικά την νίκη τους αλλά και να δυσκολέψει τον αντίπαλο, η γνώση η οποία προέρχεται από την ανάλυση κάθε κίνησης από μία ας πούμε συνάρτηση οφέλους, η οποία λειτουργεί ως εξής: για κάθε κίνηση που γίνεται από έναν παίκτη ο αντίπαλος του μελετά από τις διαθέσιμες κινήσεις που έχει ποια θα τον συμφέρει συνολικά στην πορεία του στο παιχνίδι, αυτό αξιολογείται με βάση τους κανόνες που προαναφέραμε. Επομένως βλέπουμε ότι με κάποιο τρόπο ο ένας παίκτης ενώ έχει τις δικές του πιθανές κινήσεις πρέπει να «δει» τα αποτελέσματα που θα προκαλέσει η καθεμία στην πορεία του παιχνιδιού.

Αυτό τον αλγόριθμο αναζήτησης τον ονομάζουμε **MiniMax** και εκτελείται με χρήση δέντρου αντιστρόφου DFS. Προσπελαύνουμε το δέντρο ξεκινώντας από το φύλλο, το οποίο πάντα έχει τον ρόλο τελικής κατάστασης, εφαρμόζοντας τη συνάρτηση οφέλους που μελετάει το όφελος της κίνησης κάθε παίκτη . Έπειτα ξεκινάμε να «ανεβαίνουμε» και βρίσκουμε πρώτα μεταξύ των φύλλων ποιο ωφελεί παραπάνω τον αντίπαλο ,ύστερα ανεβαίνοντας ένα επίπεδο στο δέντρο, ο αλγόριθμος επιλέγει το φύλλο που εξυπηρετεί λιγότερο τον αντίπαλο ,ωφελώντας εμάς. Αν κάνουμε χρήση DFS δημιουργούνται εξαρχής όλα τα φύλλα και γίνεται ο έλεγχος. Αυτό γίνεται καθώς η μορφή του συγκεκριμένου αλγορίθμου σε δέντρο, θέτει ως ρίζα την συνάρτηση

max και στο επόμενο επίπεδο την συνάρτηση min, η οποία αξιολογεί τα φύλλα με βάση την σκοπιά του αντιπάλου. Σε αντίθεση η max συνάρτηση επιλέγει από τα αποτελέσματα της αξιολόγησης της min συνάρτησης το χειρότερο αποτέλεσμα για τον αντίπαλο μας, το οποίο είναι το καλύτερο για εμάς. Αριθμητικά βέβαια αν αντιστοιχίσουμε την αξιολόγηση της min συνάρτησης να γίνεται με αρνητικούς αριθμούς (όσο μικρότερο αριθμό έχει ως αποτέλεσμα τόσο πιο ιδανική είναι η κίνηση για τον συγκεκριμένο παίκτη) γίνεται η παραδοχή ότι όντως η συνάρτηση max επιλέγει το μέγιστο των αποτελεσμάτων από την συνάρτηση min. Ουσιαστικά με τον συγκεκριμένο αλγόριθμο επιλέγουμε κάθε φορά τις κινήσεις που μας συμφέρουν περισσότερο, δηλαδή αυτές με τις μέγιστες ή ελάχιστες, αντίστοιχα, τιμές Minimax. Αν όμως κάποιος από τους δύο δεν επιλέξει με βάση αυτές, αλλά τις αντίθετες π.χ., τότε το παιχνίδι θα καταλήξει με την νίκη μας, ή γενικότερα το προβάδισμα μας σε σχέση με τον αντίπαλο.

Για να αναγνωρίσουμε λοιπόν την πολυπλοκότητα ενός παιχνιδιού αντιπάλων, λαμβάνοντας υπόψη και τα μειονεκτήματα ενός παίκτη που δεν κάνει την κίνηση του βάση μίας συνάρτησης οφέλους, θα μπορούσαμε να εξετάσουμε την περίπτωση ο ένα αντίπαλος να είναι η μηχανή, η οποία αποφασίζει με βάση μίας συνάρτησης οφέλους και ακολουθεί όλα τα βήματα του αλγορίθμου που προαναφέραμε, και ενός ανθρώπου, ο οποίος δεν λειτουργεί με κάποια προκαθορισμένη συνάρτηση οφέλους, δηλαδή του αλγορίθμου, αλλά παίρνει τις αποφάσεις του από προσωπικές γνώσεις. Μία τέτοια μορφή παιχνιδιού αντιπάλων αναπτύξαμε για το παιχνίδι Reversi. Η υλοποίηση από σκοπιά του παίκτη ήταν αρκετά απλή, καθώς το μόνο που είχαμε να λάβουμε υπόψη είναι να του δίνουμε κάθε φορά τις πιθανές κινήσεις που μπορεί να κάνει και με βάση την είσοδο του να κάνουμε τις αλλαγές στον πίνακα. Αντίθετα από σκοπιά μηχανής, υλοποιήσαμε τον παραπάνω αλγόριθμο MiniMax, ο οποίος προσφέρει την πλήρη αξιολόγηση κάθε τελικής κατάστασης, δηλαδή σε κάθε μία κίνηση του παίκτη όταν έρχεται η σειρά της μηχανής, εξετάζει από τον τρέχοντα πίνακα μία ροή κινήσεων που καταλήγει σε τελική κατάσταση, οι οποίες αποθηκεύονται στα φύλλα. Με τον τρόπο αυτό καταφέρνουμε να εξάγουμε την πληροφορία της κίνησης που θα κάνουμε μετά, η οποία πάρθηκε αξιολογώντας τις επιπτώσεις που θα έχει στον αντίπαλο και τις συνέπειες, προς συμφέρον μας, που θα έχει για το τελικό παιχνίδι. Λόγω της δημιουργίας αυτού του δέντρου για ένα παιχνίδι σαν το δικό μας με εκτενές πλήθος πιθανών κινήσεων αλλά και αρκετών κανόνων που εφαρμόζονται στα φύλλα ,μπορούμε να καταλάβουμε ότι η εξαγωγή των αποτελεσμάτων δεν θα είναι τόσο άμεση, αν θέλουμε να καλύψουμε το σύνολο των παραπάνω. Γι' αυτό και στην αρχή ζητάμε από τον παίκτη να επιλέξει ο ίδιος τον βαθμό δυσκολίας του παιχνιδιού, που ουσιαστικά κατευθύνει το πόσες πιθανές κινήσεις θα αξιολογήσει ο αλγόριθμος, δηλαδή απλοϊκά το ύψος του δέντρου. Αυτό που διευκόλυνε την ανάπτυξη του παιχνιδιού είναι ότι δεν πρόκειται για παιχνίδι που έχει κάποιο τυχαίο παράγοντα, επειδή παίζουν μόνο 2 παίκτες και λόγω του μεγάλου εύρους πληροφοριών, που βρήκαμε από διάφορες πηγές, όσον αφορά τους «κανόνες» που πρέπει να πληροί κάθε φορά η κίνηση για να είναι συμφέρουσα για τον αντίστοιχο παίκτη, καταλήγουμε στο ότι πρόκειται για ένα απλοποιημένο παιχνίδι αντιπάλων, το οποίο καλύπτεται πλήρως με την χρήση του συγκεκριμένου αλγορίθμου.

Συνοψίζοντας, το παιχνίδι Reversi που αναπτύξαμε αναφέρεται σε δύο παίκτες, από τους οποίους ο ένας είναι φυσικό πρόσωπο και ο άλλος μηχανή. Τις κινήσεις κάθε φορά ο πρώτος τις αποφασίζει με δική του κρίση, ενώ ο δεύτερος με χρήση του αλγορίθμου MiniMax, ο οποίος αξιολογεί με βάση μίας συνάρτησης οφέλους που εφαρμόζει σε κάθε τελική κατάσταση, που

θεωρούμε ότι αντιστοιχεί στο αποτέλεσμα από την εκτέλεση όλων των διαθέσιμων κινήσεων που είχε, τοποθετώντας τις στα φύλλα του δέντρου. Το δέντρο δημιουργείται για την εξυπηρέτηση του αλγορίθμου, και με την συνάρτηση κόστους αξιολογεί την κατάσταση που επιφέρει η κάθε κίνηση, δηλαδή τα φύλλα. Η συνάρτηση κόστους σχηματίστηκε με βάση την διαθέσιμη πληροφορία που υπάρχει στο διαδίκτυο όσον αφορά συμφέρουσες στρατηγικές που πρέπει να ακολουθήσει κάποιος για να κερδίσει, δίνοντας σε κάθε μία από αυτές την αντίστοιχη βαρύτητα, δηλαδή όσο πιο «καλή» κίνηση είναι τόσο πιο υψηλό αριθμό θα δίνει η συνάρτηση οφέλους (διατηρώντας στο νου ότι οι βαθμοί/πόντοι της min αξιολογούνται με αρνητικούς αριθμούς, όπως εξηγήσαμε παραπάνω).

Αρχιτεκτονική παιχνιδιού

Για την δημιουργία του παιχνιδιού δημιουργήσαμε 3 βασικές κλάσεις την Player, η οποία περιλαμβάνει

- την δήλωση του παίκτη
- το χρώμα που του αντιστοιχεί και το επίπεδο που επιλέχθηκε
- την ανάλυση των μεθόδων του max και του min που χρησιμοποιεί ο υπολογιστής (αφού ο άλλος παίκτης είναι άνθρωπος)
- την Move, στην οποία έχουμε όλες τις μεθόδους/συναρτήσεις που αφορούν την κίνηση κάποιου player
- την Board, η οποία περιέχει όλες τις συναρτήσεις/μεθόδους που αναφέρονται στον τρέχοντα πίνακα κάθε φορά του παιχνιδιού, καθώς και την υλοποίηση των στρατηγικών που έχουμε επιλέξει να αναλύσουμε.

Αναλυτικά για κάθε κλάση έχουμε:

Player:

Περιλαμβάνει έναν κατασκευαστή ο οποίος ορίζει το επίπεδο του παιχνιδιού (maxDepth) καθώς και το χρώμα του παίκτη. Επίσης υλοποιώντας τις συναρτήσεις max και min ,κάνουμε εφαρμογή του αλγορίθμου MiniMax και δημιουργούμε έτσι με αντίστροφο DFS το δέντρο. Πάντα ο παίκτης επιλέγει πρώτος το χρώμα που θα έχει, δηλαδή αν θα παίξει πρώτος ή όχι (όποιος διαλέξει το μαύρο χρώμα παίζει πρώτος), το οποίο σημαίνει ότι το χρώμα που θα αντιστοιχεί στον υπολογιστή είναι πάντα το αντίθετο από του παίκτη. Έχουμε συνδέσει την συνάρτηση **max** να υλοποιείται για τον παίκτη με το **μαύρο χρώμα** ενώ η **min** για αυτόν με το **άσπρο** χρώμα. Οι συγκεκριμένες συναρτήσεις καλούνται από την εφαρμογή μας (την main) αφότου έχει επιλέξει ο παίκτης χρώμα. Έτσι με βάση το χρώμα που απέμεινε καλείται η αντίστοιχη συνάρτηση για τον υπολογιστή, μέχρι να τελειώσει το παιχνίδι. Στο εσωτερικό τώρα των συγκεκριμένων συναρτήσεων, αφού χρησιμοποιούμε DFS χωρίς οπισθοδρόμηση, παράγουμε όλα τα πιθανά παιδιά με την βοήθεια της συνάρτησης που βρίσκεται στην κλάση Board.getchildren(playerLetter); . Με τον όρο παιδί εννοούμε μία από τις τελικές καταστάσεις που εξάγουμε από το τρέχοντα πίνακα, δηλαδή η υλοποίηση μίας από τις διαθέσιμες κινήσεις που αντιστοιχούν στον παίκτη. Τότε για κάθε παιδί, δηλαδή τελική κατάσταση, θέλουμε να δούμε τι κινήσεις μπορεί να κάνει ο άλλος παίκτης, εξαιτίας αυτής, επομένως καλούμε την αντίθετη συνάρτηση (για την max την min και το αντίστροφο). Εξετάζοντας όλα τα παιδιά, τα

οποία εξαρτώνται από το επίπεδο που έχει οριστεί ,δηλαδή για τα πιο δύσκολα επίπεδα χτίζονται παραπάνω επίπεδα (μεγαλύτερο δέντρο σε βάθος), καθώς λαμβάνει υπόψη πέρα από τις άμεσες επιδράσεις της κάθε κίνησης φύλλου στον άλλον παίκτη, τις δικές του πιθανές κινήσεις μετά από αυτό. Γενικότερα υπάρχει μία πιο συνολική εικόνα για παραπάνω από ένα βήμα παρακάτω αν αυξήσουμε το μέγιστο βάθος του αλγορίθμου MiniMax . Το αν είναι βέλτιστο ένα παιδί γίνεται με εσωτερική σύγκριση αρχικά του ενός αρχικοποιημένου με ακραίες τιμή MIN_VALUE για την max και ενός MAX_VALUE για την min, ώστε να είναι σίγουρο ότι θα κρατήσουμε την αξιολόγηση του πρώτου παιδιού για να την συγκρίνουμε με την επόμενη, διατηρώντας από τις δύο την καλύτερη κτλ. , μέχρις ότου να ελέγξουμε όλα τα παιδιά. Είναι σημαντικό να αναφέρουμε ότι σύγκριση γίνεται μεταξύ αντικειμένων Move για αυτό και πρώτα βάζουμε τις τιμές MIN_VALUE και MAX_VALUE αντίστοιχα ως ορίσματα. Αν πάμε να καλέσουμε οποιαδήποτε από τις δύο συναρτήσεις και δούμε ότι δεν μπορεί να εξελιχθεί άλλο το δέντρο, έχοντας φτάσει στο μέγιστο μήκος που του έχει οριστεί από το επίπεδο ή το παιχνίδι έχει ολοκληρωθεί, τότε με βάση την τελευταία κίνηση (πληροφορία που εξάγουμε από την κλάση Board) αξιολογούμε την κατάσταση του board. Αυτό γίνεται όπως προαναφέραμε στο εσωτερικό της συνάρτησης max όπου καλείται η συνάρτηση min για κάθε παιδί, αυξάνοντας κάθε φορά το μήκος του δέντρου της min. Αυτό ορίζει το «πόσο έξυπνος» γίνεται ο υπολογιστής με βάση το πόσες κινήσεις μπροστά αξιολογεί. Επομένως, με το που φτάσει στο ορισμένο βάθος αξιολογείται η τελευταία κίνηση, και τότε γίνεται η σύγκριση της τιμής αυτής με άλλες των άλλων παιδιών, όπως ήδη αναφέραμε. Επίσης στην περίπτωση που ο αντίπαλος δεν έχει διαθέσιμες κινήσεις, το οποίο γίνεται όταν η κίνηση που επιστρέφεται από την αξιολόγηση του παιδιού έχει την default αξία, για να μπορέσει ο υπολογιστής να πάρει μια απόφαση (αφού δεν υπάρχουν κινήσεις αντιπάλου για να αξιολογήσει),γυρίζουμε στο ακριβώς προηγούμενο επίπεδο δέντρου, το οποίο αντιστοιχεί στις δικές του κινήσεις και διαλέγει μία από αυτές. Με άλλα λόγια αν ο αντίπαλος δεν έχει διαθέσιμες κινήσεις στα παραγόμενα παιδιά, το δέντρο δεν αυξάνει επίπεδο, άρα το επίπεδο/βάθος του δέντρου είναι 1 υποχρεώνοντας έτσι την αξιολόγηση των παιδιών του υπολογιστή με βάση την συνάρτηση που του αντιστοιχεί (min/max ανάλογα το χρώμα του). Η εξήγηση που δώσαμε παραπάνω είναι μία συνοπτική εξήγηση για την υλοποίηση σε πρώτο επίπεδο με βάθος 2, δηλαδή μία κλήση της max και μία της min, το οποίο σημαίνει ότι υλοποιούμε την max και την min από μία φορά. Για τα υψηλότερα επίπεδα ακολουθούμε μία συλλογιστική από κάτω προς τα πάνω όπου ανεβάζουμε το επίπεδο μέχρι να μπορεί να βρει κάποια διαθέσιμη κίνηση από την πλευρά του αντιπάλου, ώστε να την αξιολογήσει. Γενικότερα καλούμε (εξαιτίας του μικρού βαθμού επιπέδων έχουμε δηλώσει ανά επίπεδο (όσο «ανεβαίνουμε») την κλήση της συνάρτησης μας, ανάλογα με maxDepth, που είναι το αρχικό επίπεδο) την συνάρτηση που μας αντιστοιχεί, δηλαδή για την max ξανά καλούμε την max, λόγω του ότι βρήκαμε ότι ο αντίπαλος δεν μπορεί να κάνει κίνηση με βάση το παιδί που εξετάζουμε, που σημαίνει ότι ανεβαίνουμε επίπεδο στο οποίο θα είναι η αντίθετη από το επίπεδο που ήμασταν συνάρτηση (αφού το δέντρο πάει max-min-max-..). Όμοια εκτελούνται τα πράγματα και για την συνάρτηση min, αξιολογώντας τα παιδιά της με βάση την συνάρτηση max.

Board: Η συγκεκριμένη κλάση είναι ιδιαίτερα σημαντική όσον αφορά την εκτέλεση όλης της διαδικασίας του αλγορίθμου MiniMax. Μέσα σε αυτή, πέρα από τον κατασκευαστή της στον οποίο δημιουργείται ο πίνακας, με την αρχική του κατάσταση όπως μας δίνεται από το παιχνίδι, έχουμε και συναρτήσεις βοηθητικές καθώς και μεθόδους που εξυπηρετούν στην λειτουργία του παιχνιδιού. Αυτές είναι

- η συνάρτηση που εκτυπώνει τον πίνακα (print) όπως και διάφορα getters τιμών
- η μέθοδος που εκτελεί/εφαρμόζει την κίνηση που έγινε στον πίνακα (makeMove)
- η μεθόδος που δηλώνει την τελευταία κίνηση (getLastMove)
- η μέθοδος που δηλώνει τον τελευταίο παίκτη που έπαιξε (πολύ χρήσιμες πληροφορίες για την εκτέλεση του αλγορίθμου και της αξιολόγησης) (getLastPlayer)
- η συνάρτηση που επιστρέφει τις διαθέσιμες κινήσεις για ορισμένο παίκτη (getavailableMoves)
- συναρτήσεις που αντιστοιχούν στις διάφορες στρατηγικές που υλοποιούνται (ex.getStableDiscs)- ορισμένες υλοποιούνται στο εσωτερικό της evaluate ενώ για άλλες δημιουργήσαμε καινούριες συναρτήσεις γιατί ορισμένες εξυπηρετούν και στην λειτουργία άλλων μεθόδων/συναρτήσεων
- η συνάρτηση που επιστρέφει την αξιολόγηση των διαθέσιμων κινήσεων (evaluate)
- η μέθοδος switcher που κάνει με βάση τους κανόνες την αλλαγή του χρώματος των ήδη τοποθετημένων «δίσκων»
- η συνάρτηση getChildren μία απαραίτητη συνάρτηση για την λειτουργία του αλγορίθμου
- η συνάρτηση isTerminal που ελέγχει αν βρισκόμαστε στο τέλος του παιχνιδιού. Αυτό γίνεται είτε αν έχουν καταληφθεί όλες οι θέσεις, άρα αυτός με τους περισσότερους «δίσκους» κερδίζει, είτε αν έχουν γίνει 2 συνεχόμενες παραλείψεις (skip), 1 από τον παίκτη και αμέσως μετά 1 από τον υπολογιστή, ή αν έχουμε ενός μόνο χρώματος δίσκους στον πίνακα (σπάνιο).

Αναλυτικότερα αξίζει να σχολιάσουμε, πέρα των στρατηγικών που θα περιγράψουμε παρακάτω, τις συναρτήσεις evaluate, getChildren, getavailableMoves.

getavailableMoves(playerLetter):

Η συγκεκριμένη συνάρτηση δέχεται ως όρισμα το χρώμα για το οποίο θα εμφανίσει τις διαθέσιμες κινήσεις. Με την λογική αυτή καλείται από την εφαρμογή μας κάθε φορά ,είτε για να παρουσιάσει στον παίκτη τις δυνατές κινήσεις που μπορεί να κάνει, είτε στο εσωτερικό της getChildren, η οποία ουσιαστικά δίνει τις κινήσεις που δημιουργούν τα φύλλα στο δέντρο που εκφράζει τον αλγόριθμο MiniMax. Για να βρούμε τις διαθέσιμες κινήσεις ελέγχουμε με βάση τα frontiers του αντιπάλου(του άλλου χρώματος), τα οποία είναι οι «δίσκοι» όπου έχουν σε μία από τις 8 κατευθύνσεις που μπορούν να τους «επιτεθεί» κάποιος κενό. Με βάση αυτά τα στοιχεία ανιχνεύουμε σε ποιες από αυτές τις κενές θέσεις μας επιτρέπεται να προσθέσουμε «δίσκο» δικού μας χρώματος. Αυτό γίνεται με βάση τον έλεγχο ανά θέση των 8 κατευθύνσεων για δικό μας δίσκο σε σχέση με του αντιπάλου και της αντίθετης φοράς κατεύθυνσης αναζήτησης κενής θέσης, ώστε να θεωρηθεί διαθέσιμη κίνηση.

getChildren(letter):

Η συγκεκριμένη συνάρτηση επιστρέφει τις πιθανές κινήσεις του υπολογιστή ή του player, που παίρνουν ουσιαστικά ρόλο παιδιού καθώς με βάση ό,τι έχουμε πει αξιολογούμε κάθε μία από αυτές σαν να εκτελέσθηκε, με σκοπό να κρατήσουμε την βέλτιστη ώστε να κερδίσουμε το παιχνίδι, εξετάζοντας από την ανάλογη σκοπιά κάθε φορά (σχετικό με max ή min).

Evaluate():

Είναι μία συνάρτηση που αξιολογεί κάθε φορά τον τρέχοντα πίνακα με βάση τις στρατηγικές, για τις οποίες αναζητήσαμε από αρκετές πηγές και συμπεριλάβαμε όλες τις κοινές, που είχαν αναφερθεί σε κάθε πηγή, οι οποίες ήταν τα corners, mobility, stable discs, frontier, parity , maximum disc strategy καθώς και κάποιες μη δημοφιλείς όπως το ποιος θα καταλάβει το κέντρο του πίνακα (είτε το αρχικό (2x2) είτε το γενικότερο που είναι τα εσωτερικά 4x4), wedges, positioning και unstable edges. Σε ορισμένες στρατηγικές μεταβάλλεται η αξία τους σε σχέση με την φάση του παιχνιδιού (θεωρούμε αρχή του παιχνιδιού< 20 κινήσεων). Τους πόντους τους θέσαμε με βάση την σπουδαιότητα κάθε στρατηγικής, όπως την εκλάβαμε από τις πηγές μας, με χρήση σταθερών αριθμών για προσαύξηση της προσφοράς της κάθε κίνησης σε κάθε παίκτη. Όπως έχουμε ήδη αναφέρει τους πόντους του παίκτη που έχει τους άσπρους δίσκους, τους προσαυξάνουμε με αρνητικούς αριθμούς με την λογική που ακολουθεί η MiniMax.

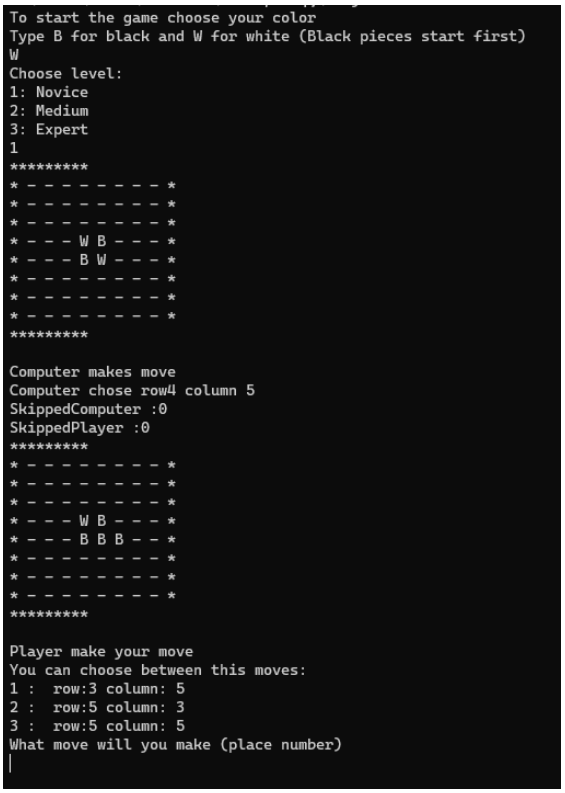
ΠΕΡΙΓΡΑΦΗ

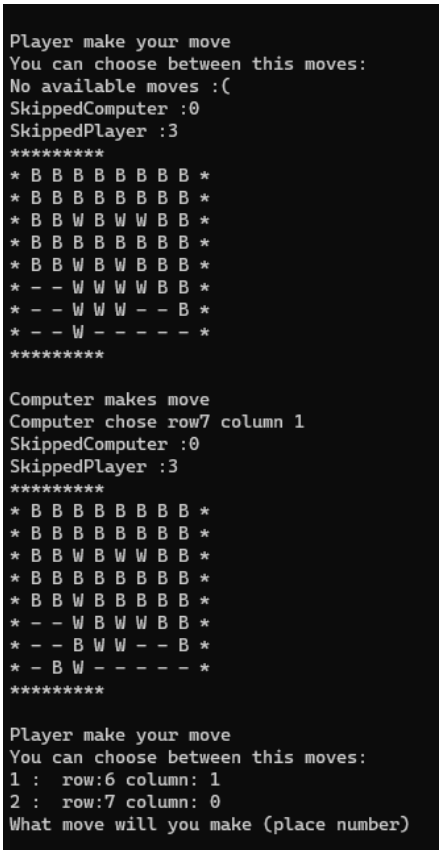
```
To start the game choose your color
Type B for black and W for white (Black pieces start first)
B
Choose level:
1: Novice
2: Medium
3: Expert
1
*********
* - - - - - - - - *
* - - - - - - - - *
* - - - - - - - - *
* - - - W B - - - *
* - - - B W - - - *
* - - - - - - - - *
* - - - - - - - - *
* - - - - - - - - *
*********

Player make your move
You can choose between this moves:
1 :  row:2 column: 3
2 :  row:3 column: 2
3 :  row:5 column: 4
4 :  row:4 column: 5
What move will you make (place number)
```

- Όπως βλέπουμε και στην εικόνα , μέσω κατάλληλων μηνυμάτων ζητάμε την επιλογή του χρώματος που θέλει ο παίκτης καθώς και το level δυσκολίας. (Warning: πάντα ο black παίζει πρώτος). Εμφανίζουμε το board με τις αρχικές κινήσεις και σε αυτή την περίπτωση όπου ο παίκτης έχει επιλέξει να είναι Black , θα παίξει και πρώτος, εκτυπώνοντας του τις διαθέσιμες κινήσεις.

- Σε αυτή την περίπτωση ο παίκτης έχει επιλέξει το χρώμα White και γι' αυτό το προβάδισμα το έχει ο computer. Ο computer επιλέγει κίνηση και έτσι μας εκτυπώνεται το board, περιλαμβάνοντας την επιλεγμένη του κίνηση και έχοντας κάνει switch τα πιόνια που χρειάζεται.

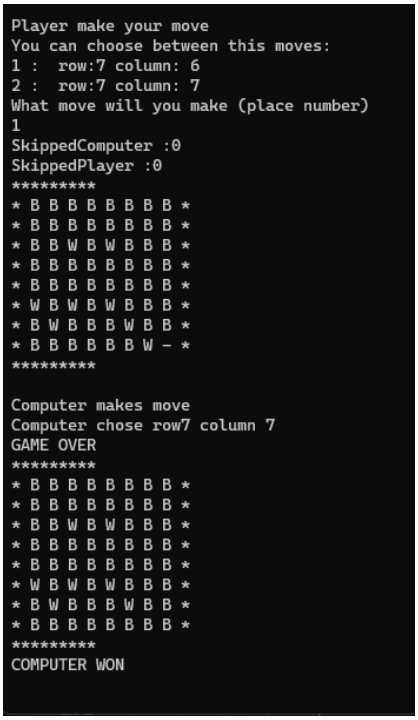


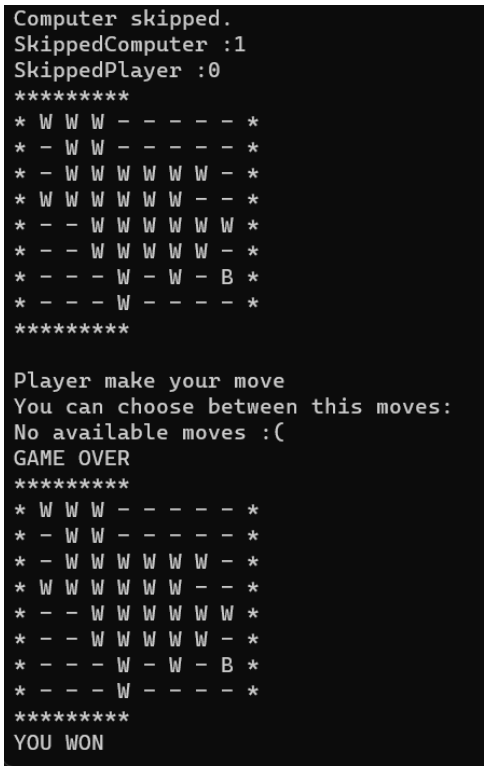


- Εδώ πρόκειται για την περίπτωση όπου ο παίκτης δεν έχει διαθέσιμες κινήσεις . Έτσι λοιπόν εμείς τυπώνουμε το κατάλληλο μήνυμα καθώς και αυξάνουμε την μεταβλητή skip του player κατά 1. Στην συνέχεια εμφανίζουμε το ίδιο board με πριν αφού δεν έχει αλλάξει κάτι και η σειρά είναι του υπολογιστή.

ΠΕΡΙΠΤΩΣΕΙΣ ΤΕΡΜΑΤΙΣΜΟΥ ΠΑΙΧΝΙΔΙΟΥ

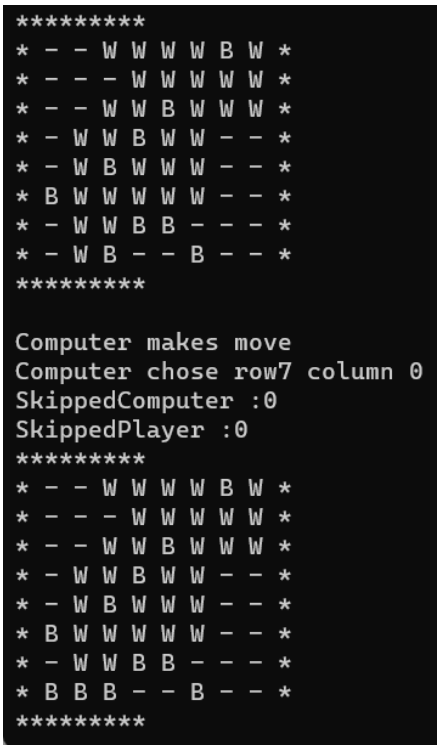
1. Το παιχνίδι έχει τερματίσει καθώς το board έχει γεμίσει όλες τις θέσεις του. Στο τέλος τυπώνουμε πάντα τον νικητή ο οποίος κρίνεται με βάση το πλήθος των πιονιών τους. Ο παίκτης με τα περισσότερα πιόνια είναι και ο νικητής.





2. Έχουν γίνει 2 συνεχόμενα skip(ένα του υπολογιστή και ένα του παίκτη). Το παιχνίδι τερματίζει πριν γεμίσει το board . Όπως βλέπουμε και από την μεταβλητή , ο computer κάνει skip καθώς δεν έχει διαθέσιμες κινήσεις και όταν είναι η σειρά του παίκτη δεν έχει ούτε αυτός κίνηση(No available moves)επομένως αλλάζει και η δική του μεταβλητή. Το παιχνίδι τελειώνει με νικητή τον player,καθώς ως white έχει περισσότερα πιόνια.

ΣΤΡΑΤΗΓΙΚΗ



• Σε αυτή την περίπτωση μπορούμε να παρατηρήσουμε πως ο υπολογιστής επιλέγει μια έξυπνη κίνηση με βάση την στρατηγική corner nodes. Επιλέγει δηλαδή να τοποθετήσει να στη γωνία το πιόνι του αντί για οποιαδήποτε άλλη κίνηση ,καθώς η γωνία προσφέρει περισσότερους πόντους και μπορεί να περικυκλώσει πιο εύκολα τον αντίπαλο αφού δύσκολα πλέον μπορούν να του πάρουν αυτή τη θέση.

Στρατηγικές

- **Stable Discs** :Υλοποιείται με την συνάρτηση getStableDiscs(int playerLetter) ,η οποία εκτιμά ποιοι δίσκοι ενός παίκτη είναι σταθεροί ,δηλαδή ο αντίπαλος δεν θα μπορεί να τους γυρίσει για την υπόλοιπη διάρκεια του παιχνιδιού και επιστρέφει στην evaluate μία λίστα με αυτούς. Αρχικά μέσα στην λίστα προστίθενται σίγουρα οι γωνίες αν έχουν καταληφθεί από τον παίκτη και ξεκινώντας από αυτές ελέγχουμε αν και οι 5 περιμετρικές θέσεις κάθε δίσκου προς εξέταση έχουν το ίδιο χρώμα με αυτό του παίκτη και αν ναι τότε ο δίσκος αυτός προστίθεται στην λίστα. Στο τέλος βγάζουμε από την λίστα τα διπλότυπα και γυρίζοντας στην evaluate δίνουμε πόντους σε κάθε παίκτη ανάλογα με το πλήθος των σταθερών δίσκων που διαθέτει (όσο μεγαλύτερο πλήθος τόσο περισσότεροι πόντοι).
- **Frontiers** : Υλοποιείται με την συνάρτηση getFrontierSquares(int playerLetter) ,η οποία μας επιστρέφει μία λίστα με τους δίσκους που βρίσκονται στο μέτωπο για κάθε παίκτη, δηλαδή τους δίσκους που έχουν δίπλα τους έστω και μία κενή θέση σε οποιαδήποτε από τις 8 κατευθύνσεις. Αφαιρούμε από την λίστα αυτή τους σταθερούς δίσκους διότι δίπλα σε αυτούς εξ ορισμού ο αντίπαλος δεν μπορεί να τοποθετήσει δικό του δίσκο ,λόγω του ότι είναι αδύνατο να τους γυρίσει και να τους αλλάξει χρώμα. Έτσι όσο μικρότερο είναι το μέτωπο ενός παίκτη τόσες λιγότερες κινήσεις μπορεί ο αντίπαλος του να κάνει. Οπότε στην evaluate καλώντας αυτήν την συνάρτηση συγκρίνουμε τα

πλήθη των frontiers και για τους δύο παίκτες και εκείνος με το μικρότερο πλήθος παίρνει πόντους ενώ ο αντίπαλος χάνει πόντους.

- **Wedges (Σφήνες)** : Υλοποιείται με τις συναρτήσεις getUpWedges ,getDownWedges , getLeftWedges, getRightWedges και κάθε μία από αυτές αντιστοιχεί σε μία από τις 4 άκρες του ταμπλό(πχ πρώτη σειρά- getUpWedges, πρώτη στήλη- getLeftWedges κ.ο.κ.). Σε κάθε άκρη υπολογίζουμε το πλήθος των κενών θέσεων που υπάρχουν μεταξύ δύο δίσκων του ίδιου χρώματος (προφανώς αν υπάρχει ένας μόνο δίσκος σε μια άκρη δεν υπολογίζουμε πλήθος) και κάθε πλήθος το προσθέτουμε σε μία λίστα .Κρατάμε μία λίστα για τους άσπρους(whkeno) και μία για τους μαύρους δίσκους(blkeno). Ύστερα γυρίζουμε στην evaluate μία λίστα που περιέχει αυτές τις δύο λίστες, τις διατρέχουμε ξεχωριστά και ελέγχουμε για κάθε πλήθος αν είναι μονό ή ζυγό .Για κάθε ζυγό (μη μηδενικό) πλήθος ο παίκτης κερδίζει πόντους και για κάθε μονό χάνει. Τα μονά πλήθη δεν είναι ευνοϊκά γιατί παίζοντας σε αυτά ο αντίπαλος μπορεί να <<μπει σφήνα>> και να καταλάβει την άκρη του ταμπλό.
- **Κατάληψη κέντρου του πίνακα** (είτε 2X2 είτε 4X4): Σε αυτήν την περίπτωση απλά συγκρίνουμε τα πλήθη των άσπρων και μαύρων δίσκων που βρίσκονται σε αυτές τις περιοχές του ταμπλό και σε κάθε παίκτη προστίθενται πόντοι ανάλογα με το πλήθος των δίσκων που έχουν(όσο μεγαλύτερο τόσο το καλύτερο). Είναι σημαντικές οι περιοχές αυτές γιατί μας παρέχουν περισσότερες μελλοντικές διαθέσιμες κινήσεις άρα και περισσότερους δικούς μας δίσκους τοποθετημένους στο ταμπλό.
- **Κατάληψη Γωνιών** : Ελέγχουμε ποιος παίκτης έχει καταλάβει τις περισσότερες γωνίες του ταμπλό και του δίνουμε πόντους ανάλογα με το πλήθος των γωνιών που έχει στην κατοχή του. Αυτή είναι από τις σημαντικότερες στρατηγικές γιατί αρχικά αν καταληφθεί μια γωνία γίνεται σταθερός δίσκος και δεν μπορεί να αλλάξει ξανά χρώμα και δεύτερον μπορούμε έτσι να καταλάβουμε μία ολόκληρη άκρη ή μια διαγώνιο του ταμπλό κερδίζοντας πλεονέκτημα έναντι του αντιπάλου και τελικά το ίδιο το παιχνίδι.
- **Στρατηγική Περισσότερων Δίσκων**: Συγκρίνουμε το πλήθος των άσπρων και μαύρων δίσκων και ανάλογα με την φάση του παιχνιδιού στην οποία βρισκόμαστε ,που υπολογίζεται με το πλήθος των συνολικών γεμάτων θέσεων στο ταμπλό , αποδίδουμε πόντους στους παίκτες. Μόνο στην τελευταία φάση ,δηλαδή να υπάρχουν παραπάνω από 58 δίσκοι στο ταμπλό, το να έχει κάποιος περισσότερους δίσκους πάνω στο ταμπλό μετράει θετικά για το σκορ του και αρνητικά για το σκορ του αντιπάλου.
- **Mobility:** Υλοποιείται με την συνάρτηση getAvailableMoves(int playerLetter) ,η οποία μας επιστρέφει την λίστα με τις νόμιμες διαθέσιμες κινήσεις για κάθε παίκτη. Συγκρίνουμε τα 2 πλήθη και ο παίκτης με το μεγαλύτερο πλήθος κερδίζει πόντους ενώ ο αντίπαλος χάνει.Στόχος είναι να έχουμε εμείς περισσότερες κινήσεις από τον αντίπαλο και ο αντίπαλος να έχει όσες λιγότερες γίνεται ώστε να αναγκαστεί να διαλέξει μια κακή κίνηση.
- **Parity**: Αυτή η στρατηγική έχει νόημα από την μέση φάση του παιχνιδιού και ύστερα έτσι ώστε να έχουν σχηματιστεί ξεχωριστές περιοχές με άδειες θέσεις και εμείς ορίσαμε αυτήν την φάση ως το συνολικό πλήθος των δίσκων στο ταμπλό να είναι μεγαλύτερο ίσο του 20. Επειδή αξιολογούμε κάθε φορά ένα φύλλο άρα την τελευταία κίνηση που έγινε θεωρητικά στο board παίρνουμε τις συντεταγμένες της και υπολογίζουμε το πλήθος των κενών θέσεων που υπάρχουν γύρω της ,στις 8

κατευθύνσεις. Αν το πλήθος πλέον είναι ζυγό σημαίνει ότι η κίνηση έγινε αρχικά σε μονό πλήθος κενών άρα ήταν μια καλή κίνηση και ο παίκτης κερδίζει πόντους και ο αντίπαλός του χάνει, αντίθετα αν το πλήθος πλέον είναι μονό ο παίκτης χάνει πόντους και ο αντίπαλος του κερδίζει.

- **Positioning**: Σε αυτήν την στρατηγική πρέπει να αποφεύγει ο παίκτης να παίξει σε X-square αν η γωνία που βρίσκεται δίπλα και διαγώνια σε αυτό είναι κενή και σε περίπτωση που επιλέξει αυτήν την κίνηση χάνει πόντους.
- **Unbalanced Edges:** Υλοποιείται με την συνάρτηση unbalancedEdges() μέσα στην οποία ελέγχουμε για κάθε άκρη από τις 4 αν οι δίσκοι που βρίσκονται σε αυτήν είναι κεντραρισμένοι ,δηλαδή το πλήθος των κενών θέσεων εκατέρωθεν τους είναι ίδιο. Αυτή η στρατηγική χρησιμοποιείται μόνο αν σε μια άκρη υπάρχουν δίσκοι ενός χρώματος αλλιώς δεν την συμπεριλαμβάνουμε την άκρη αυτή στον υπολογισμό του σκορ. Αν μία άκρη είναι ισοζυγισμένη ο παίκτης κερδίζει πόντους αλλιώς αντίστροφα χάνει. Οι πόντοι υπολογίζονται μέσα στην συνάρτηση και τους επιστρέφουμε με μία λίστα στην evaluate ,όπου και προστίθενται στο συνολικό σκορ του κάθε παίκτη.